Reinforcement Learning: An Intuitive Introduction with Mathematics and Algorithms

Ahmed BADI
ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed

January 2026

Abstract

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative reward. Unlike supervised learning, RL does not rely on labeled input-output pairs but learns from trial and error and delayed feedback. This article provides an intuitive introduction to RL, formalizes the problem using Markov Decision Processes (MDPs), and presents core algorithms such as dynamic programming, Monte Carlo methods, temporal-difference learning, and Q-learning. We also discuss key mathematical concepts like value functions, the Bellman equation, exploration versus exploitation, and convergence. Figures and simple examples are used to make the material accessible, while equations give a solid foundation for further study.

Keywords: Reinforcement Learning, Markov Decision Process, Value Function, Bellman Equation, Dynamic Programming, Q-learning, Temporal-Difference Learning.

1 Introduction

Reinforcement Learning is about learning from interaction. An agent repeatedly observes the state of an environment, chooses actions, receives numerical rewards, and adapts its behavior to achieve long-term goals. [1], [2]

Unlike supervised learning, where correct labels are provided for each example, RL agents must discover which actions lead to high rewards by experimenting with the environment. Feedback is often delayed: a decision may only show its true quality many steps later. [1]

Common RL applications include:

- Game playing (chess, Go, Atari, robotics control).
- Recommendation systems (optimizing long-term engagement).
- Robotics and control (autonomous driving, manipulation).
- Operations research (inventory management, scheduling).

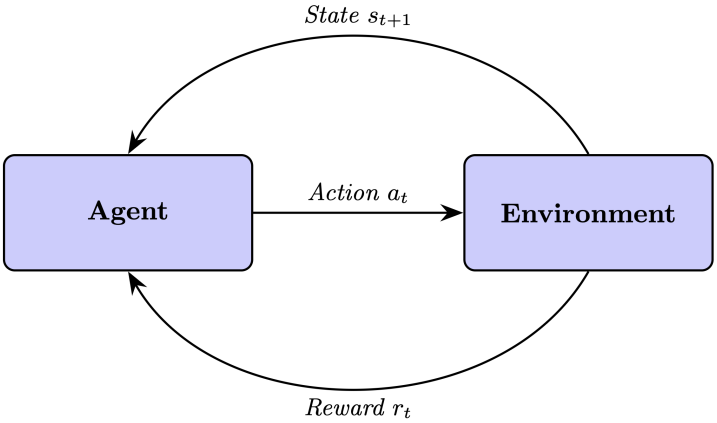

Figure 1: Basic RL loop: the agent observes the state, selects an action, receives a reward and a new state from the environment.

2 The Reinforcement Learning Framework

RL problems are typically modeled as Markov Decision Processes (MDPs). [1], [3]

2.1 Markov Decision Process (MDP)

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: set of states.
- $\mathcal{A}$: set of actions.
- $P(s' \mid s, a)$: transition probability from state s to s' when taking action a.
- $R(s, a, s')$: reward received when transitioning from s to s' via action a.
- $\gamma \in [0, 1)$: discount factor, controlling how future rewards are valued.

At each time step t, the agent:

1. Observes current state $S_t \in \mathcal{S}$.
2. Chooses action $A_t \in \mathcal{A}$ according to a policy.
3. Receives reward R_{t+1} and next state S_{t+1}.

2.2 Policy

A policy π specifies how the agent selects actions:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s).$$

π can be deterministic (always pick one action) or stochastic (sample actions with probabilities). The goal of RL is to find an optimal policy $\pi^\star$ that maximizes expected cumulative reward. [1], [3]

2.3 Return and Discounting

The *return* from time t is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}.$$

The discount factor γ controls how much future rewards matter; γ close to 1 values long-term outcomes, while lower γ focuses more on immediate rewards. [1]

3 Value Functions and the Bellman Equation

Value functions estimate how good it is for an agent to be in a given state or to take a given action. [1], [2]

3.1 State-Value Function

For a policy π, the state-value function $v_\pi(s)$ is:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,\middle|\, S_t = s \right].$$

It measures the expected return starting from state s and following policy π.

3.2 Action-Value Function

The action-value function $q_\pi(s, a)$ is:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$

It measures the expected return starting from state s, taking action a, and then following policy π.

3.3 Bellman Expectation Equations

The value functions satisfy recursive relationships known as Bellman equations. For v_π: [1]

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right].$$

Similarly, for q_π:

$$q_\pi(s, a) = \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right].$$

These equations express the value of a state (or state-action pair) in terms of immediate reward plus the discounted value of the next state.

3.4 Optimal Value Functions

An optimal policy $\pi^\star$ yields optimal value functions $v_\star(s)$ and $q_\star(s, a)$:

$$v_\star(s) = \max_\pi v_\pi(s), \quad q_\star(s, a) = \max_\pi q_\pi(s, a).$$

They satisfy the Bellman optimality equations:

$$v_\star(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\star(s') \right],$$

$$q_\star(s, a) = \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma \max_{a'} q_\star(s', a') \right].$$

4 Dynamic Programming Methods

When the MDP model P and R is known and the state space is small, dynamic programming (DP) can be used to compute optimal policies exactly. [1]

4.1 Policy Evaluation

Given a fixed policy π, policy evaluation computes v_π by iteratively applying the Bellman expectation equation:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right].$$

Starting from an initial guess v_0, this iteration converges to v_π.

4.2 Policy Improvement

Once v_π is known, we can improve the policy by acting greedily with respect to v_π:

$$\pi_{\text{new}}(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right].$$

This produces a strictly better policy unless π is already optimal. [1], [4]

4.3 Policy Iteration

Policy iteration alternates policy evaluation and improvement: [1], [4]

1. Initialize π_0 arbitrarily.
2. **Policy evaluation:** Compute v_{π_k}.
3. **Policy improvement:** Compute π_{k+1} greedily from v_{π_k}.
4. Repeat until policies stop changing.

4.4 Value Iteration

Value iteration combines evaluation and improvement into a single update:

$$v_{k+1}(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right].$$

This is equivalent to repeatedly applying the Bellman optimality operator and converges to $v_\star$. [1], [5]

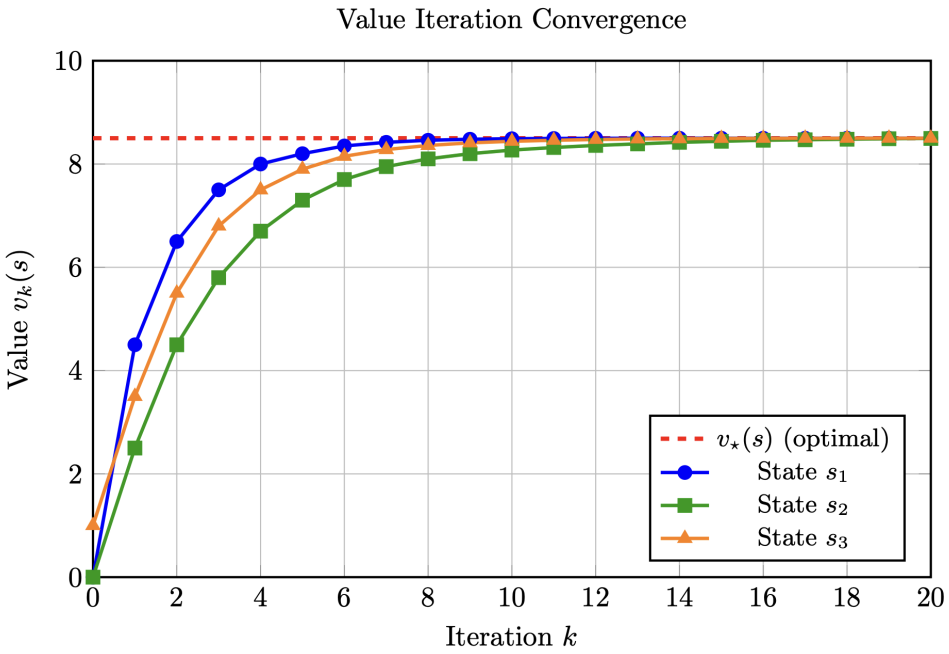

Figure 2: Example of value iteration converging to the optimal value function over iterations.

5 Monte Carlo Methods

Monte Carlo (MC) methods estimate value functions from sample episodes without knowing the transition model. [1]

5.1 Monte Carlo State-Value Estimation

For a policy π, we can estimate $v_\pi(s)$ by averaging returns following visits to state s:

$$v_\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s),$$

where $G^{(i)}(s)$ is the return after the i-th occurrence of state s, and $N(s)$ is number of visits. [1]

5.2 Incremental Updates

Instead of storing all returns, we can update incrementally:

$$v_\pi(s) \leftarrow v_\pi(s) + \alpha \left(G - v_\pi(s)\right),$$

where α is a step size and G is the observed return.

Monte Carlo methods require complete episodes but are simple and work with unknown dynamics.

6 Temporal-Difference Learning

Temporal-Difference (TD) methods combine ideas from dynamic programming and Monte Carlo. They learn value functions from experience but update estimates using bootstrapping (using existing estimates). [1], [2]

6.1 TD(0) for State-Value

The simplest TD method, TD(0), updates state values after each step:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right].$$

The term

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

is the TD error, measuring surprise between predicted and updated returns. [1]

6.2 Advantages of TD Methods

- Learn online, step by step.
- Do not require full episodes (can work in continuing tasks).
- Often more data-efficient than pure MC methods.

7 Q-Learning: Model-Free Control

Q-learning is a popular model-free RL algorithm that learns the optimal action-value function $Q_\star(s, a)$ directly from experience. [6], [7]

7.1 Q-Learning Update Rule

Q-learning maintains a table $Q(s, a)$ and updates it using:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right].$$

This is a TD update for the action-value function, using the greedy action in the next state. [7], [8]

7.2 Policy from Q-Values

Given Q-values, a greedy policy chooses:

$$\pi(s) = \arg\max_a Q(s, a).$$

To encourage exploration, ϵ-greedy policies are used:

- With probability ϵ, choose a random action.
- With probability $1 - \epsilon$, choose the action with highest Q-value.

7.3 Exploration vs Exploitation

RL must balance:

- **Exploration:** trying new actions to discover their effects.
- **Exploitation:** choosing actions known to yield high rewards.

Strategies include ϵ-greedy, softmax action selection, and upper confidence bounds (UCB). [1], [6]

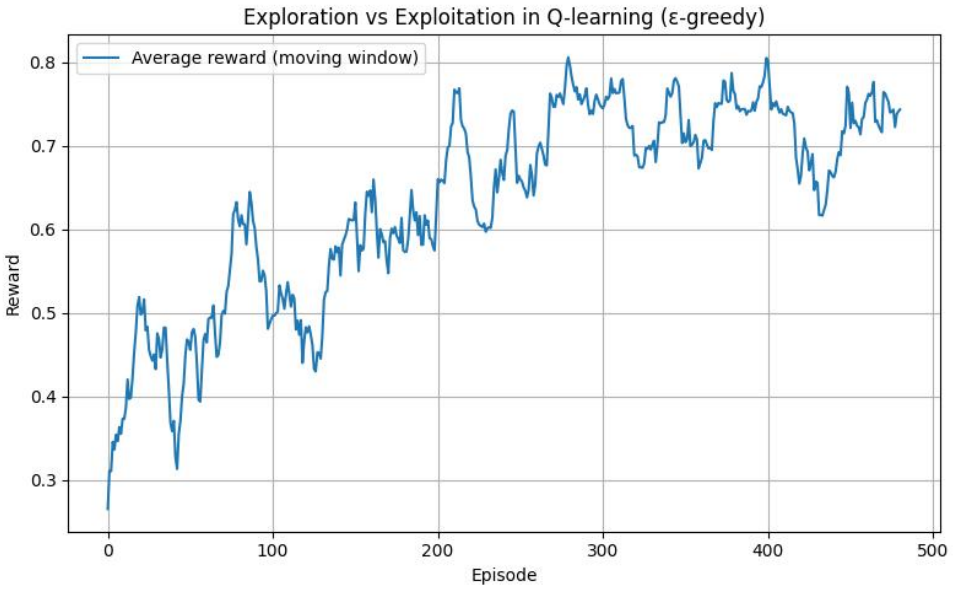

Figure 3: Example learning curve: cumulative reward improving over episodes for Q-learning.

8 Summary of Core Tabular Algorithms

Table 1 provides a compact comparison of core tabular RL methods.

Method	**What it estimates**	**Needs model?**	**Typical use**
Policy Evaluation	$v_\pi(s)$ for fixed π	Yes (full MDP)	Analyze given policy
Policy Iteration	Optimal $\pi^\star$ via v_π	Yes	Exact solution for small MDPs
Value Iteration	$v_\star(s)$ and $\pi^\star$	Yes	Exact planning
Monte Carlo	$v_\pi(s)$ or $q_\pi(s,a)$ from episodes	No	Episodic tasks, unknown dynamics
TD(0)	$v_\pi(s)$ online via TD error	No	Online prediction
Q-Learning	$Q_\star(s,a)$ via off-policy TD control	No	Model-free control

Table 1: Key tabular reinforcement learning methods.

9 Stochastic Policies and Policy Gradient (Brief Overview)

So far, focus has been on value-based methods (value functions, Q-learning). Another family of methods directly parameterizes the policy and optimizes its parameters using gradient ascent on expected return. [1]

9.1 Parameterized Policy

Let $\pi_\theta(a \mid s)$ be a differentiable policy with parameters θ. The objective is to maximize the expected return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_0].$$

Policy gradient methods estimate $\nabla_\theta J(\theta)$ and update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

9.2 REINFORCE (Monte Carlo Policy Gradient)

A simple algorithm, REINFORCE, uses the gradient estimator: [1]

$$\nabla_\theta J(\theta) \approx \sum_t \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, G_t.$$

This connects RL to deep learning when policies are neural networks.

10 Deep Reinforcement Learning (High-Level)

Deep RL combines RL algorithms with deep neural networks to handle large or continuous state spaces. [9]

Examples:

- Deep Q-Networks (DQN): approximates Q-values with CNNs (Atari games).
- Policy gradient and actor–critic methods with neural networks (PPO, A3C).

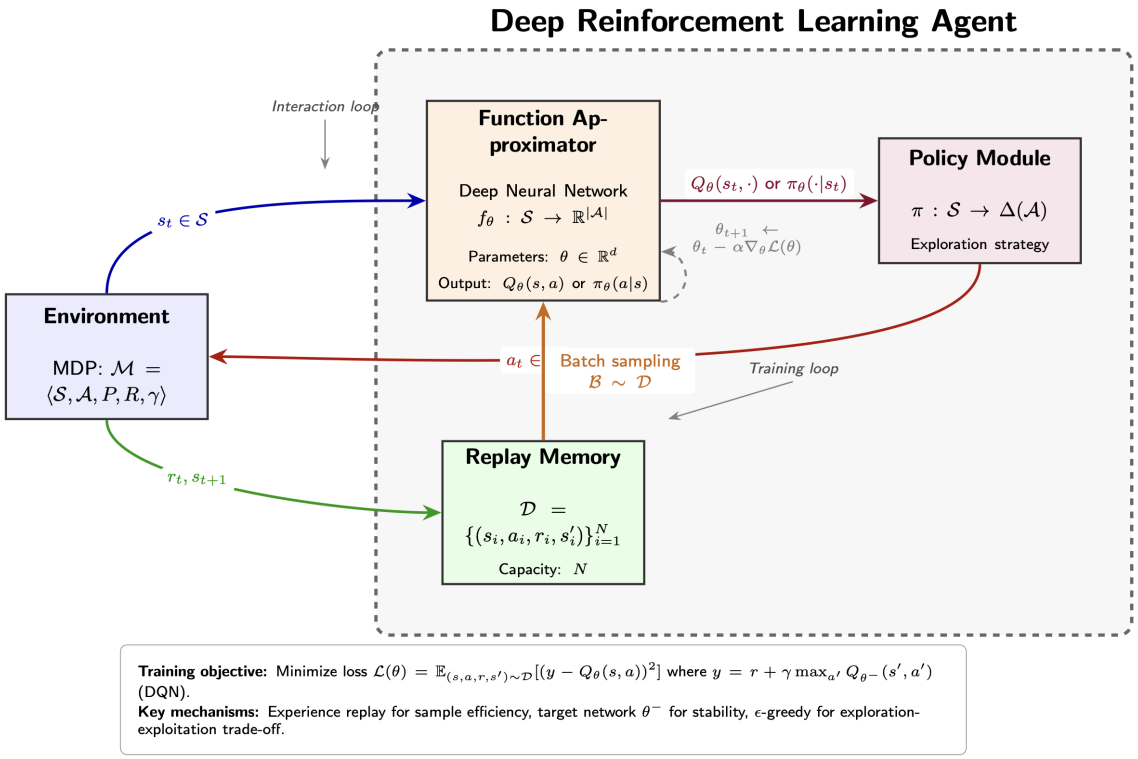

Figure 4: High-level deep RL architecture: neural network approximates value or policy, trained from interaction with environment.

11 Practical Considerations and Challenges

RL in real systems faces several difficulties: [1], [2]

- **Sample efficiency:** Many environment interactions may be needed.
- **Exploration:** Balancing exploration and exploitation safely.
- **Credit assignment:** Identifying which actions in the past caused current rewards.
- **Function approximation:** Generalizing across large or continuous state spaces.

Good practices include:

- Start with small tabular problems (gridworld, bandits) to build intuition.
- Carefully tune learning rate α, discount γ, and exploration parameters.
- Use baselines and variance-reduction techniques in policy gradients.

12 Conclusion

Reinforcement Learning provides a powerful framework for learning to act under uncertainty through trial and error. In this article, we have:

- Introduced the RL problem and MDP formulation.
- Defined value functions and Bellman equations.
- Presented dynamic programming, Monte Carlo, TD learning, and Q-learning.
- Briefly discussed policy gradient and deep RL.

This foundation prepares the way for more advanced topics such as actor–critic methods, eligibility traces, function approximation, and multi-agent reinforcement learning. [1], [9]

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[2] R. Enjeeneer, *Notes on reinforcement learning: An introduction (2nd edition)*, `https://enjeeneer.io/sutton_and_barto/rl_notes.pdf`, 2021.

[3] Wikipedia contributors, *Reinforcement learning*, `https://en.wikipedia.org/wiki/Reinforcement_learning`, 2024.

[4] Gibberblot, *Policy iteration — mastering reinforcement learning*, `https://gibberblot.github.io/rl-notes/single-agent/policy-iteration.html`, 2022.

[5] Stack Overflow, *What is the difference between value iteration and policy iteration?* `https://stackoverflow.com/questions/37370015/what-is-the-difference-between-value-iteration-and-policy-iteration`, 2021.

[6] Hugging Face, *Introducing q-learning*, `https://huggingface.co/learn/deep-rl-course/unit2/q-learning`, 2023.

[7] DataCamp, *Q-learning python tutorial*, `https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial`, 2022.

[8] Simplilearn, *Important terms in q-learning*, `https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning`, 2024.

[9] Y. Li, "A survey on deep reinforcement learning," *arXiv preprint arXiv:1812.05551*, 2018.